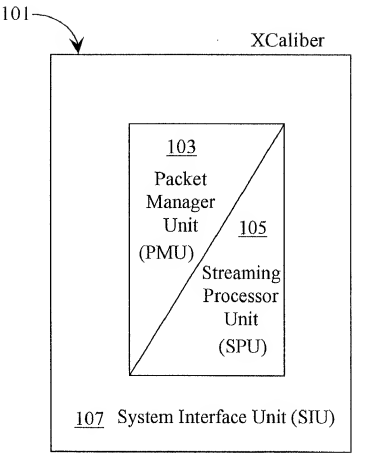


Fig. 1

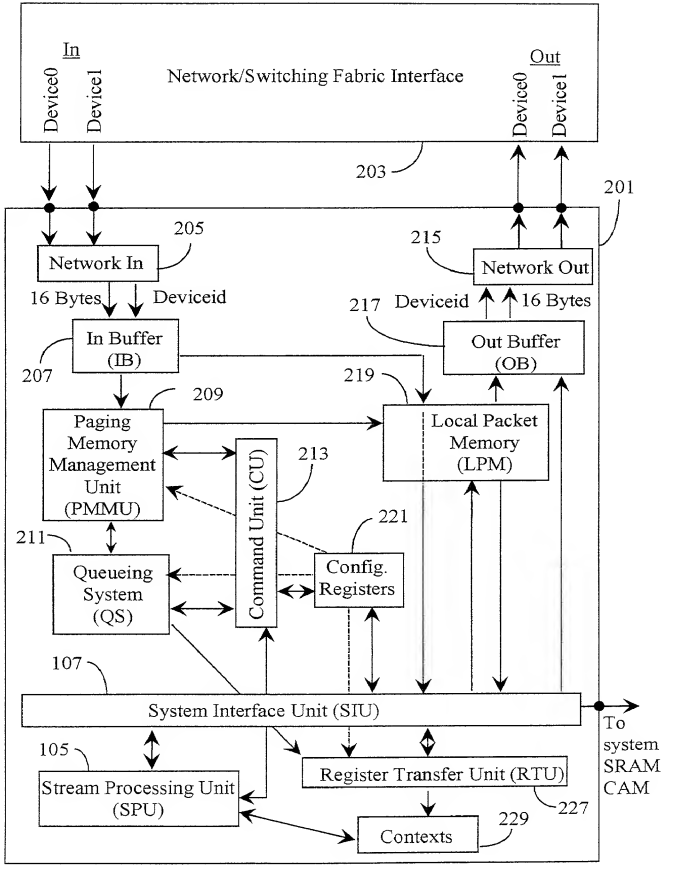


Fig. 2

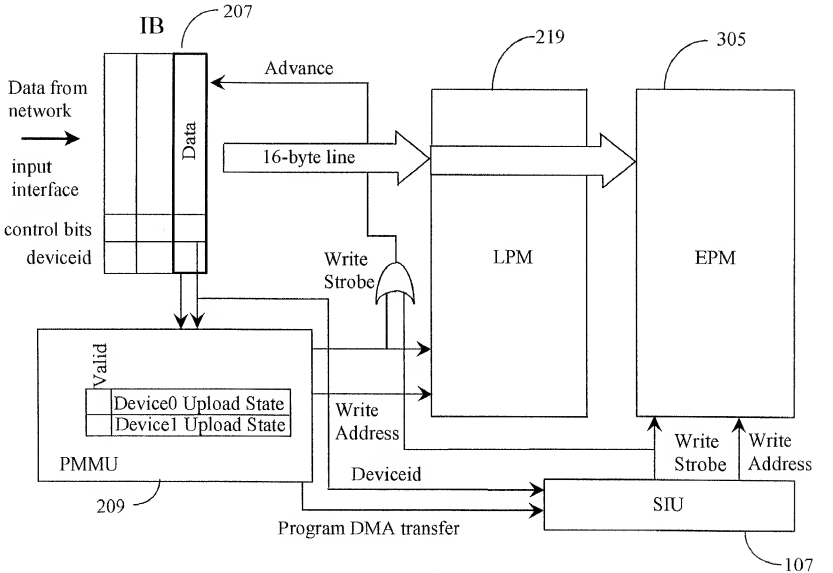


Fig. 3

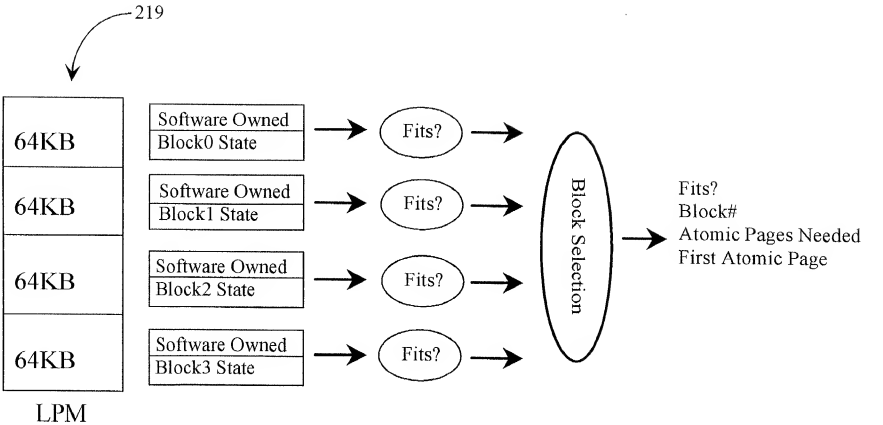


Fig. 4a

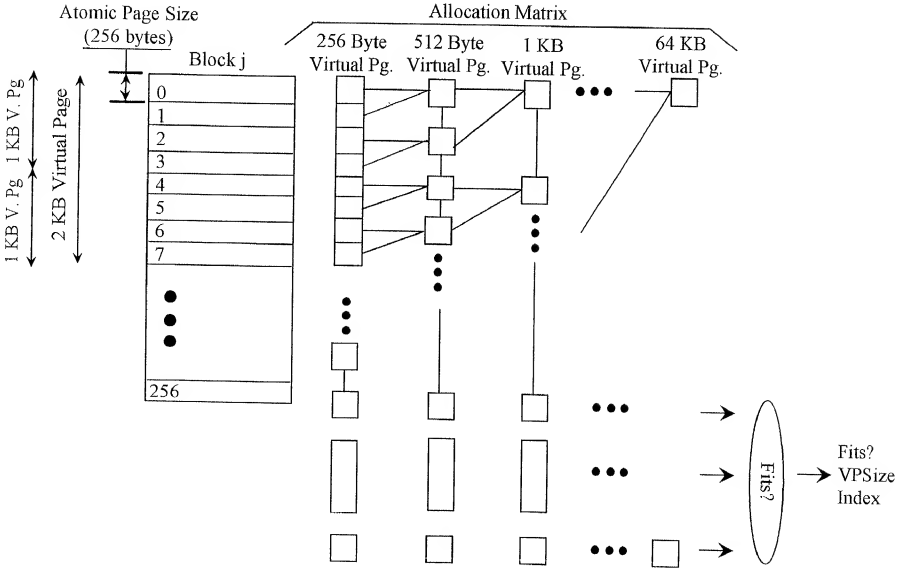


Fig. 4b

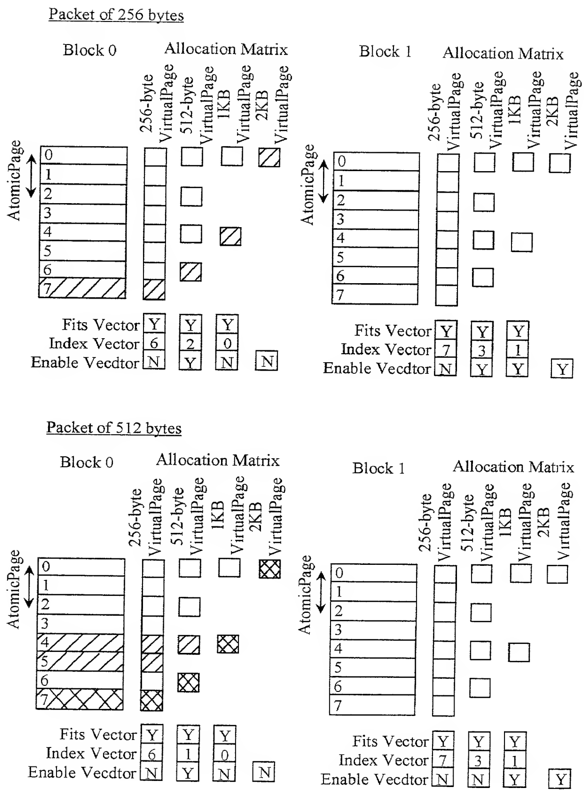

Packet of 256 bytes
Block 0
Allocation Matrix
256-byte VirtualPage
512-byte VirtualPage
1KB VirtualPage
2KB VirtualPage
AtomicPage
0
1
2
3
4
5
6
7
Fits Vector Y Y Y
Index Vector 6 2 0
Enable Vecdtor N Y N N
Block 1
Allocation Matrix
256-byte VirtualPage
512-byte VirtualPage
1KB VirtualPage
2KB VirtualPage
AtomicPage
0
1
2
3
4
5
6
7
Fits Vector Y Y Y
Index Vector 7 3 1
Enable Vecdtor N Y Y Y
Packet of 512 bytes
Block 0
Allocation Matrix
256-byte VirtualPage
512-byte VirtualPage
1KB VirtualPage
2KB VirtualPage
AtomicPage
0
1
2
3
4
5
6
7
Fits Vector Y Y Y
Index Vector 6 1 0
Enable Vecdtor N Y N N
Block 1
Allocation Matrix
256-byte VirtualPage
512-byte VirtualPage
1KB VirtualPage
2KB VirtualPage
AtomicPage
0
1
2
3
4
5
6
7
Fits Vector Y Y Y
Index Vector 7 3 1
Enable Vecdtor N N Y Y

Fig. 5a

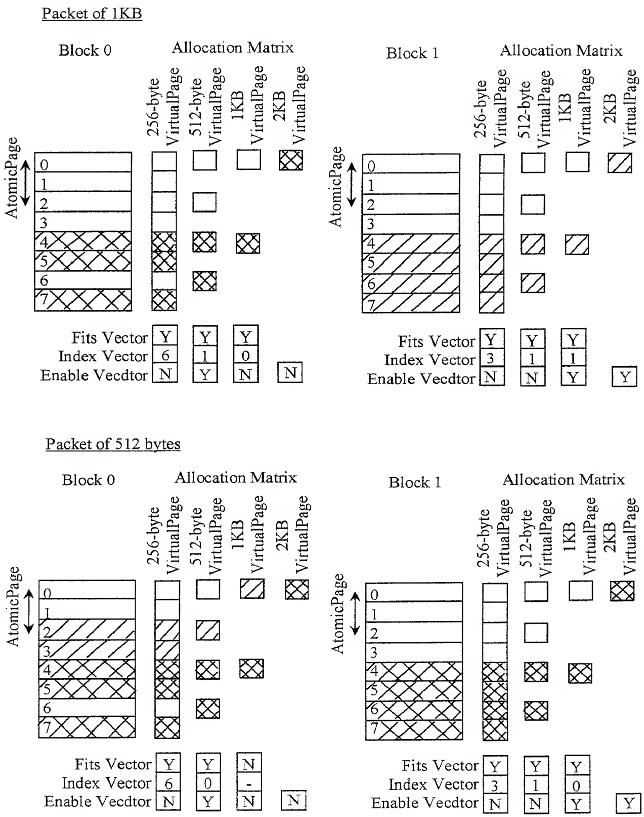

Packet of 1KB
Block 0
Allocation Matrix
256-byte VirtualPage
512-byte VirtualPage
1KB VirtualPage
2KB VirtualPage
AtomicPage
0
1
2
3
4
5
6
7
Fits Vector
Index Vector
Enable Vecdtor
Y
6
N
Y
1
Y
Y
0
N
N
Block 1
Allocation Matrix
256-byte VirtualPage
512-byte VirtualPage
1KB VirtualPage
2KB VirtualPage
AtomicPage
0
1
2
3
4
5
6
7
Fits Vector
Index Vector
Enable Vecdtor
Y
3
N
Y
1
N
Y
1
Y
Y
Packet of 512 bytes
Block 0
Allocation Matrix
256-byte VirtualPage
512-byte VirtualPage
1KB VirtualPage
2KB VirtualPage
AtomicPage
0
1
2
3
4
5
6
7
Fits Vector
Index Vector
Enable Vecdtor
Y
6
N
Y
0
Y
N
-
N
N
Block 1
Allocation Matrix
256-byte VirtualPage
512-byte VirtualPage
1KB VirtualPage
2KB VirtualPage
AtomicPage
0
1
2
3
4
5
6
7
Fits Vector
Index Vector
Enable Vecdtor
Y
3
N
Y
1
N
Y
0
Y
Y

Fig. 5b

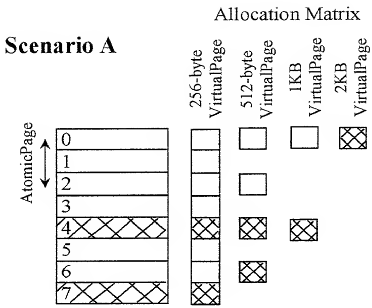

Allocation Matrix
Scenario A
256-byte VirtualPage
512-byte VirtualPage
1KB VirtualPage
2KB VirtualPage
AtomicPage
0
1
2
3
4
5
6
7

Fig. 6a

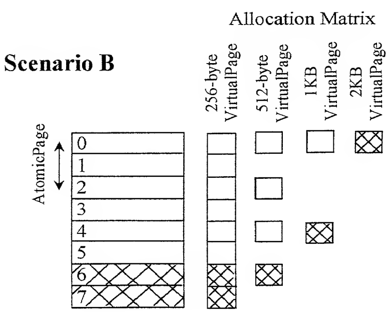

Allocation Matrix
Scenario B
256-byte VirtualPage
512-byte VirtualPage
1KB VirtualPage
2KB VirtualPage
AtomicPage
0
1
2
3
4
5
6
7

Fig. 6b

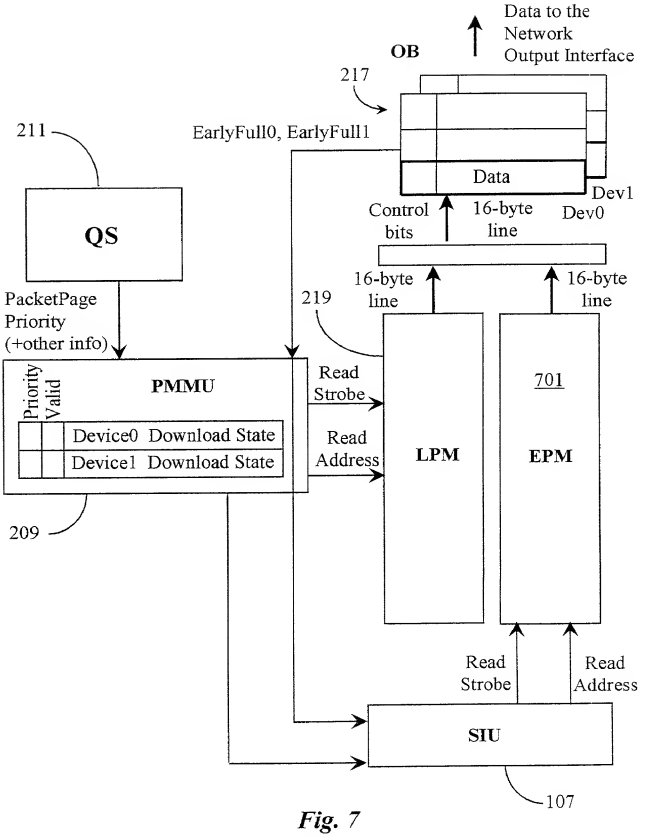
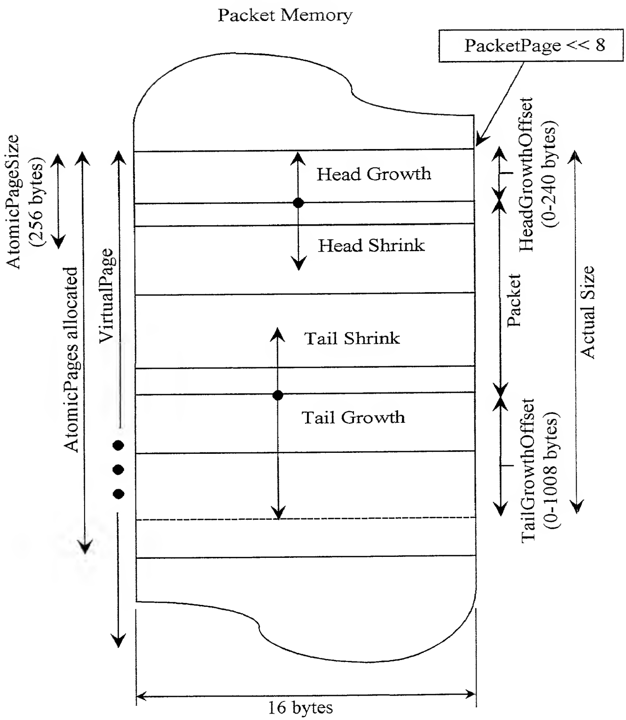

Data to the
Network
Output Interface
OB
217
211
EarlyFull0, EarlyFull1
Data
Dev1
Dev0
Control
bits
16-byte
line
QS
16-byte
line
16-byte
line
PacketPage
Priority
(+other info)
219
Priority
Valid
PMMU
Read
Strobe
701
Device0 Download State
Device1 Download State
Read
Address
LPM
EPM
209
Read
Strobe
Read
Address
SIU
107

Fig. 7



Packet Memory
PacketPage << 8
AtomicPageSize (256 bytes)
AtomicPages allocated
VirtualPage
Head Growth
Head Shrink
Tail Shrink
Tail Growth
HeadGrowthOffset (0-240 bytes)
Packet
TailGrowthOffset (0-1008 bytes)
Actual Size
16 bytes

Fig. 8

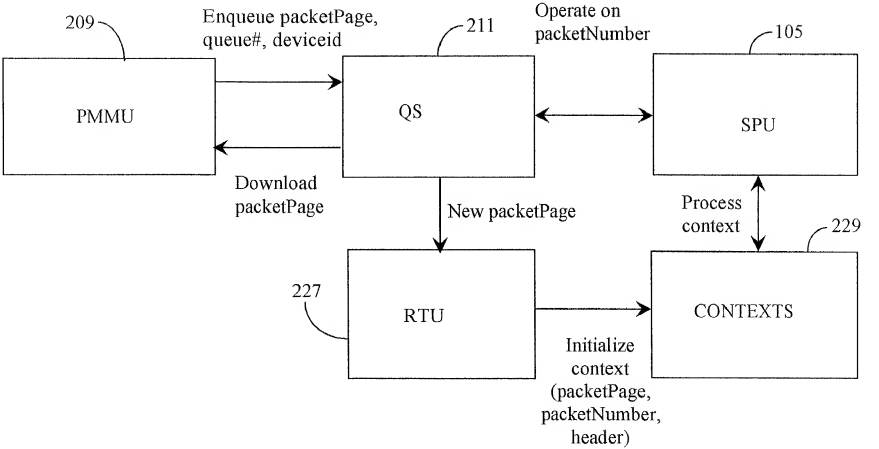


Fig. 9

Priority Clusters	# clusters	# queues / cluster	Queues in each cluster
0	1	32	{0,...,31}
1	2	16	{0,...,15},...,{16,...,31}
2	4	8	{0,...,7},...,{24,...,31}
3	8	4	{0,...,3},...,{28,...,31}
4	16	2	{0,1},{2,3},...,{30,31}
5	32	1	{0},{1},,...,{31}

Fig. 10 Clustering of queues

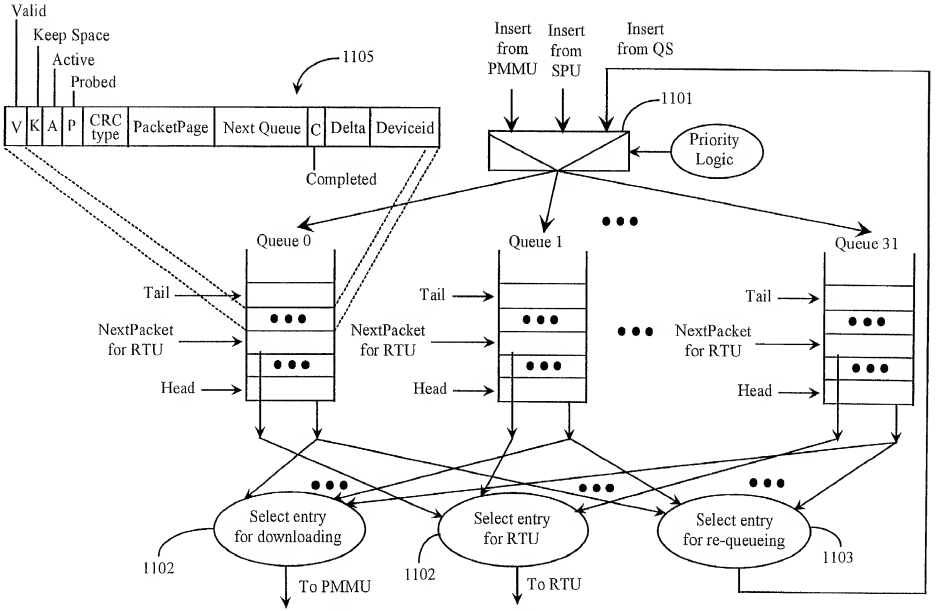


Fig. 11 Generic Queueing Architecture

InboundDeviceid Field	Outbound Device Identifier
0	Inbound Device Identifier
1	*Not Used*
2	0
3	1

Fig. 12

PriorityClusters	# clusters	RTU Priority
0	1	Queue#>>5 or Cluster# (i.e. always 0)
1	2	Queue#>>4 or Cluster#
2	4	Queue#>>3 or Cluster#
3	8	Queue#>>2 or Cluster#
4	16	Queue#>>2 or Cluster#>>1
5	32	Queue#>>2 or Cluster#>>2

Fig. 13

A	C	P	State of the Packet
0	0	-	*Never*
0	1	-	Packet is completed (could have been previously probed or not)
1	0	-	Packet is being processed by the SPU (can be probed or not)
1	1	-	*Never*
0	0	0	Packet is being processed by the SPU. This state may happen after a MoveAndReactivate operation on a not-probed packet, or after the packet is inserted by the PMMU (i.e. a new packet)
0	0	1	Packet is not being processed by the SPU. This state may hapen after a MoveAndReactivate operation on a probed packet.
0	1	-	*Never*
1	-	-	*Never*

Fig. 14

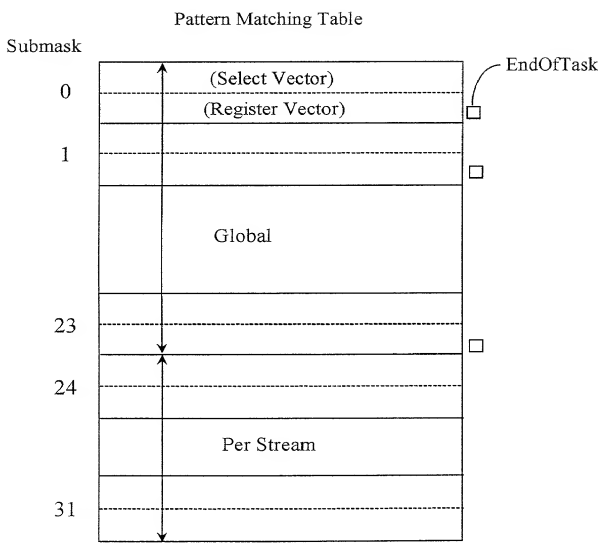

Pattern Matching Table
Submask
EndOfTask
(Select Vector)
0
(Register Vector)
1
Global
23
24
Per Stream
31

Fig. 15

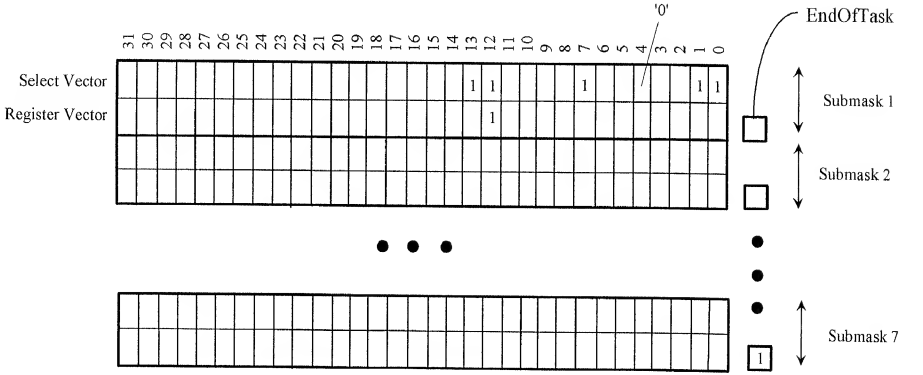
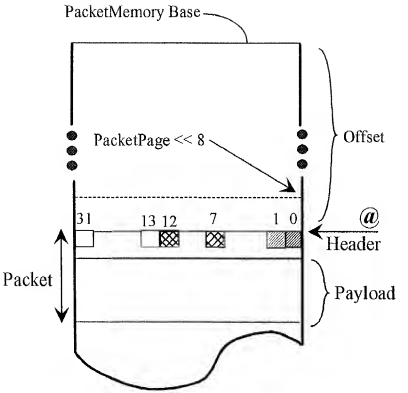


Fig. 16



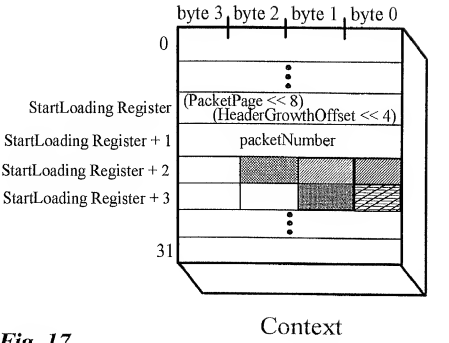


Fig. 17

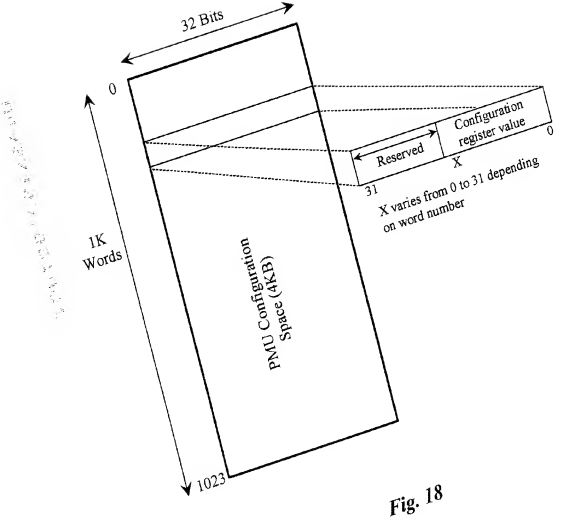

32 Bits
0
1K
Words
1023
PMU Configuration
Space (4KB)
Reserved
Configuration
register value
31
X
0
X varies from 0 to 31 depending
on word number

Fig. 18

Word #	Configuration Register Name	Block Affected
0-7	PreloadMaskNumber	RTU
5-63	*Reserved*	
64-111	PatternMatchingTable (Select and Register Vectors)	
112	*Reserved*	
448	PatternMatchingTable (EndOfMask bits)	
449	*Reserved*	
450	PacketAvailableButNoContextPriorityPintEnable	
451	DefaultPacketPriority	
452-453	ContextSpecificPatternMatchingMask0	
454-467	*Reserved*	
468-469	ContextSpecificPatternMatchingMask1	
470-483	*Reserved*	
484-485	ContextSpecificPatternMatchingMask2	
486-499	*Reserved*	
500-501	ContextSpecificPatternMatchingMask3	
502-515	*Reserved*	
516-517	ContextSpecificPatternMatchingMask4	
518-531	*Reserved*	
532-533	ContextSpecificPatternMatchingMask5	
534-547	*Reserved*	
548-549	ContextSpecificPatternMatchingMask6	
550-563	*Reserved*	
564-565	ContextSpecificPatternMatchingMask7	
566-579	*Reserved*	
580	PacketAvailableButNoContextIntMapping	
581	StartLoadingRegister	
582	CodeEntryPointSpecial	
583	*Reserved*	
584	CodeEntryPoint0	
585	CodeEntryPoint1	
586	CodeEntryPoint2	
587	CodeEntryPoint3	
588	CodeEntryPoint4	
589	CodeEntryPoint5	
590	CodeEntryPoint6	
591	CodeEntryPoint7	
592	CodeEntryPoint8	
593	CodeEntryPoint9	
594	CodeEntryPoint10	
595	CodeEntryPoint11	
596	CodeEntryPoint12	

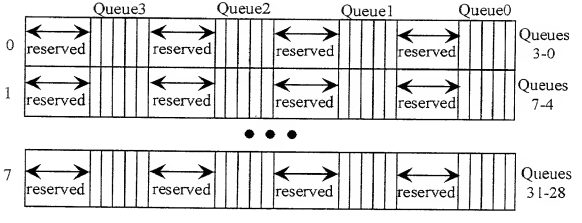
Fig.19a

597	CodeEntryPoint13	
598	CodeEntryPoint14	
599	CodeEntryPoint15	
600	CodeEntryPoint16	
601	CodeEntryPoint17	
602	CodeEntryPoint18	
603	CodeEntryPoint19	
604	CodeEntryPoint20	
605	CodeEntryPoint21	
606	CodeEntryPoint22	
607	CodeEntryPoint23	
608	CodeEntryPoint24	
609	CodeEntryPoint25	
610	CodeEntryPoint26	
611	CodeEntryPoint27	
612	CodeEntryPoint28	
613	CodeEntryPoint29	
614	CodeEntryPoint30	
615	CodeEntryPoint31	
616-767	*Reserved*	
768	Log2InputQueues	PMMU
769	HeaderGrowthOffset	
770	TailGrowthOffset	
771	PacketErrorIntEnable	
772	AutomaticPacketDropIntEnable	
773	*reserved*	
774	TimeStampEnable	
775-776	VirtualPageEnable	
777-778	*Reserved*	
779	OverflowAddress	
780	IntIfNoMoreXsizePages	
781	FirstInputQueue	
782	OverflowEnable	
783	SizeOfOverflowedPacket	
784	SoftwareOwned	
785-786	TimeCounter	
787	ClearError0	
788	ClearError1	
789-799	*Reserved*	
800-815	MaxActivePackets	QS
816-927	*Reserved*	
928	IntIfLessThanXpacketIdEntries	
929	PriorityClustering	

Fig. 19b

930-959	*Reserved*	
960	Freeze	CU
961	Reset	
962	StatusRegister	
963	BypassHooks	
964	InternalStateWrite	
965	InternalStateRead	
963-1023	*Reserved*	

Fig. 19c



PreloadMaskNumber Configuration Register

Fig. 20

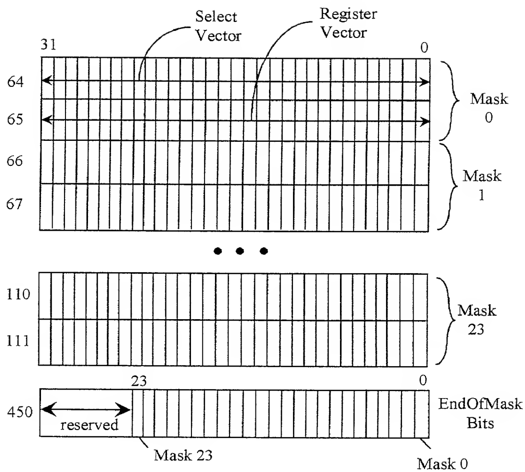


Fig. 21

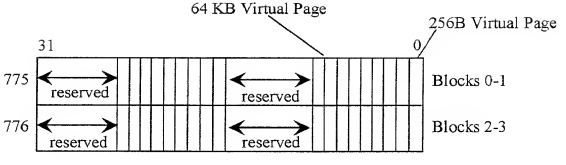

64 KB Virtual Page
256B Virtual Page
31
0
775
reserved
reserved
Blocks 0-1
776
reserved
reserved
Blocks 2-3

Fig. 22

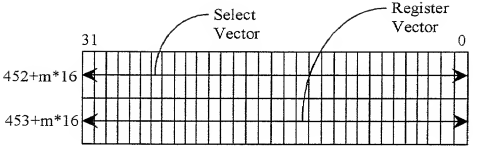


Fig. 23

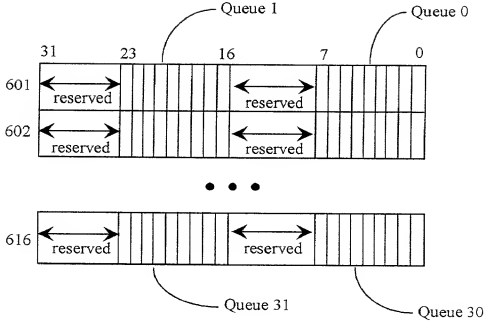


Fig. 24

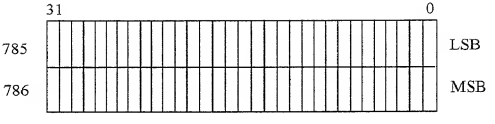


Fig. 25

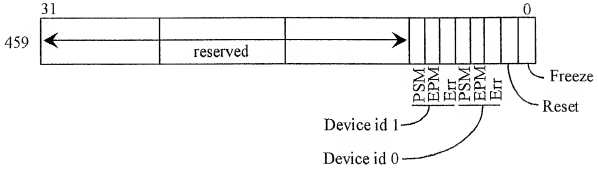


Fig. 26

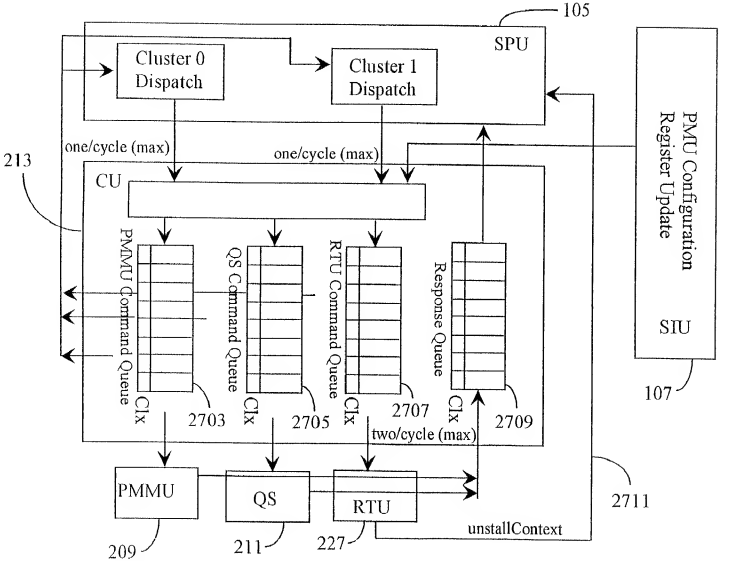


Fig. 27

Block	Command	Operand Fields	Position in Data
PMMU	0: GetSpace	Size	15..0
	1: FreeSpace	PacketPage	15..0
QS	0: InsertPacket	PacketPage	23..8
		QueueNumber	4..0
	1: ProbePacket	PacketNumber	7..0
		Set	8
	2: ExtractPacket	PacketNumber	7..0
	3: CompletePacket	PacketNumber	7..0
		Delta	17..8
		DeviceId	19..18
		CRCtype	21..20
		KeepSpace	22
	4: UpdatePacket	PacketNumber	7..0
		PacketPage	23..8
	5: MovePacket	PacketNumber	7..0
		NewQueueNumber	12..8
		Reactivate	13
	6: ProbeQueue	QueueNumber	4..0
	7: ConditionalActivate	PacketNumber	7..0
RTU	0: GetContext	N/A	N/A
	1: ReleaseContext	N/A	N/A
	2: MaskedLoad	MaskNumber	4..0
		StartRegisterNumber	9..5
		PhysicalAddress	45..10
	3: MaskedStore	MaskNumber	4..0
		StartRegisterNumber	9..5
		PhysicalAddress	45..10

Fig. 28

Block	Response To Command	Response Fields	Position in Data
PMMU	GetSpace	PacketPage	15..0
		Success	16
QSY	InsertPacket	Success	0
		PacketNumber	8..1
	ProbePacket, ProbeAndSet	Exists	0
		Completed	1
		NextQueue	6..2
		PacketPage	22..7
		DeviceId	23
		CRCtype	25..24
		Active	26
		Probed	27
		KeepSpace	28
	ProbeQueue	QueueSize	8..0
	ConditionalActivate	Success	0

Fig. 29

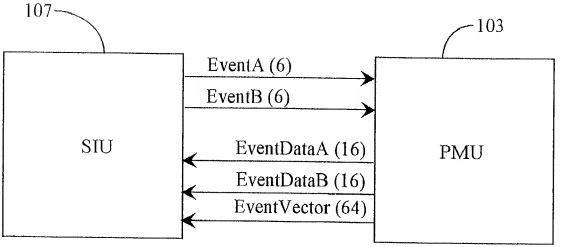


Fig. 30

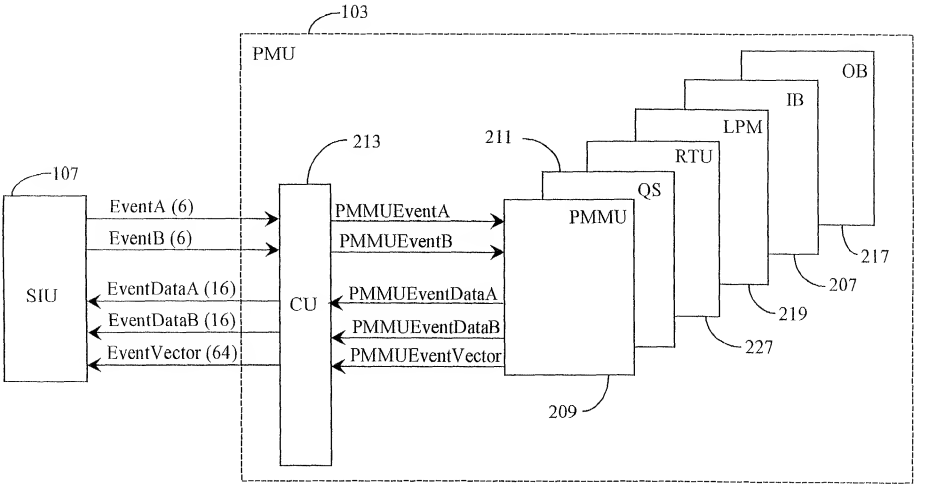


Fig. 31

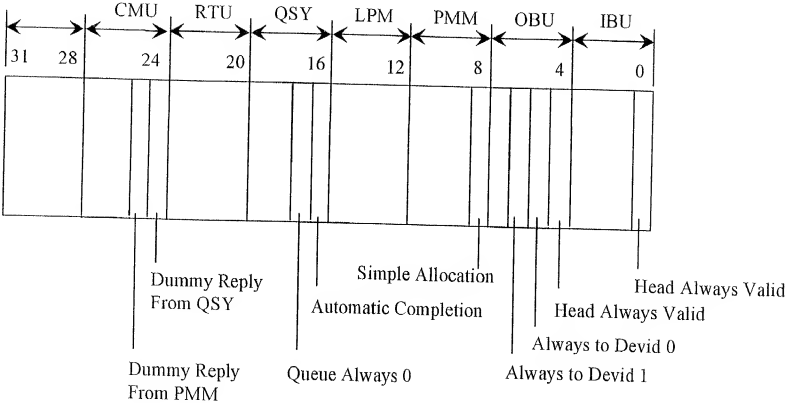


Fig. 32 ByPassHooks Configuration Register

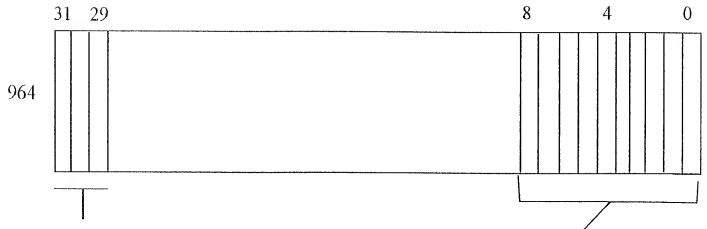


Fig. 33 InternalState Write Configuration Register

Block	Event#	Event Name	Event Data	Event Description
IB	0	*Insert*	*FreeBufferEntries (3)*	A 16-byte chunk of packet data is inserted at the tail of the IB. The event data is the number of free entries in this buffer before the insertion.
OB	1	*Insert0*	*FreeBufferEntries0 (3)*	A 16-byte chunk of packet data is inserted at the tail of the OB (device identifier 0). The event data is the number of free entries in this buffer before the insertion.
	2	*Insert1*	*FreeBufferEntries1 (3)*	A 16-byte chunk of packet data is inserted at the tail of the OB (device identifier 0). The event data is the number of free entries in this buffer before the insertion.
PMMU	3	*PacketAllocSuccess0*	*PacketSize (16)*	The PMMU successfully allocates a consecutive space in block 0 of the LPM for a packet of *PacketSize* bytes coming from the network input interface.
	4	*PacketAllocSuccess1*	*PacketSize (16)*	The PMMU successfully allocates a consecutive space in block 1 of the LPM for a packet of *PacketSize* bytes coming from the network input interface.
	5	*PacketAllocSuccess2*	*PacketSize (16)*	The PMMU successfully allocates a consecutive space in block 2 of the LPM for a packet of *PacketSize* bytes coming from the network input interface.
	6	*PacketAllocSuccess3*	*PacketSize (16)*	The PMMU successfully allocates a consecutive space in block 3 of the LPM for a packet of *PacketSize* bytes coming from the network input interface.

Fig. 34

PMMU	7	*PacketAllocFail*	*LPMfreeWords (16)*	The PMMU failed in allocating space in the LPM for a packet coming from the network input interface. The event data is the total number of words (4 bytes) free in the LPM.
	8	*PacketAllocFail*	*PacketSize (16)*	The PMMU failed in allocating space in the LPM for a packet of *PacketSize* bytes coming from the network input interface.
	9	*PacketAllocFailDrop*	*PacketSize (16)*	The PMMU failed in allocating space in the LPM for a packet of *PacketSize* bytes coming from the network input interface; the packet is dropped..
	10	*PacketAllocFailOverflow*	*PacketSize (16)*	The PMMU failed in allocating space in the LPM for a packet of *PacketSize* bytes coming from the network input interface; the packet is overflowed.
	11	*Alloc256Fail0*	*Block0FreeBytes (16)*	The allocation of a packet of 2-255 bytes failed in block 0 of LPM.
	12	*Alloc256Fail1*	*Block1FreeBytes (16)*	The allocation of a packet of 2-255 bytes failed in block 1 of LPM.
	13	*Alloc256Fail2*	*Block2FreeBytes (16)*	The allocation of a packet of 2-255 bytes failed in block 2 of LPM.
	14	*Alloc256Fail3*	*Block3FreeBytes (16)*	The allocation of a packet of 2-255 bytes failed in block 3 of LPM.
	15	*Alloc512Fail0*	*Block0FreeBytes (16)*	The allocation of a packet of 256-511 bytes failed in block 0 of LPM.
	16	*Alloc512Fail1*	*Block1FreeBytes (16)*	The allocation of a packet of 256-511 bytes failed in block 1 of LPM.
	17	*Alloc512Fail2*	*Block2FreeBytes (16)*	The allocation of a packet of 256-511 bytes failed in block 2 of LPM.

Fig. 35

PMMU	18	*Alloc512Fail3*	*Block3FreeBytes (16)*	The allocation of a packet of 256-511 bytes failed in block 3 of LPM.
	19	*Alloc1KFail0*	*Block0FreeBytes (16)*	The allocation of a packet of 512-1023 bytes failed in block 0 of LPM.
	20	*Alloc1KFail1*	*Block1FreeBytes (16)*	The allocation of a packet of 512-1023 bytes failed in block 1 of LPM.
	21	*Alloc1KFail2*	*Block2FreeBytes (16)*	The allocation of a packet of 512-1023 bytes failed in block 2 of LPM.
	22	*Alloc1KFail3*	*Block3FreeBytes (16)*	The allocation of a packet of 512-1023 bytes failed in block 3 of LPM.
	23	*Alloc2KFail0*	*Block0FreeBytes (16)*	The allocation of a packet of 1024-2047 bytes failed in block 0 of LPM.
	24	*Alloc2KFail1*	*Block1FreeBytes (16)*	The allocation of a packet of 1024-2047 bytes failed in block 0 of LPM.
	25	*Alloc2KFail2*	*Block2FreeBytes (16)*	The allocation of a packet of 1024-2047 bytes failed in block 0 of LPM.
	26	*Alloc2KFail3*	*Block3FreeBytes (16)*	The allocation of a packet of 1024-2047 bytes failed in block 0 of LPM.
	27	*Alloc4KFail0*	*Block0FreeBytes (16)*	The allocation of a packet of 2048-4095 bytes failed in block 0 of LPM.
	28	*Alloc4KFail1*	*Block1FreeBytes (16)*	The allocation of a packet of 2048-4095 bytes failed in block 1 of LPM.
	29	*Alloc4KFail2*	*Block2FreeBytes (16)*	The allocation of a packet of 2048-4095 bytes failed in block 2 of LPM.
	30	*Alloc4KFail3*	*Block3FreeBytes (16)*	The allocation of a packet of 2048-4095 bytes failed in block 3 of LPM.
	31	*Alloc16KFail0*	*Block0FreeBytes (16)*	The allocation of a packet of 4096-16383 bytes failed in block 0 of LPM.
	32	*Alloc16KFail1*	*Block1FreeBytes (16)*	The allocation of a packet of 4096-16383 bytes failed in block 1 of LPM.

Fig. 36

PMMU	33	*Alloc16KFail2*	*Block2FreeBytes (16)*	The allocation of a packet of 4096-16383 bytes failed in block 2 of LPM.
	34	*Alloc16KFail3*	*Block3FreeBytes (16)*	The allocation of a packet of 4096-16383 bytes failed in block 3 of LPM.
	35	*Alloc64KFail0*	*Block0FreeBytes (16)*	The allocation of a packet of 16384-65535 bytes failed in block 0 of LPM.
	36	*Alloc64KFail1*	*Block1FreeBytes (16)*	The allocation of a packet of 16384-65535 bytes failed in block 1 of LPM.
	37	*Alloc64KFail2*	*Block2FreeBytes (16)*	The allocation of a packet of 16384-65535 bytes failed in block 2 of LPM.
	38	*Alloc64KFail3*	*Block3FreeBytes (16)*	The allocation of a packet of 16384-65535 bytes failed in block 3 of LPM.
	39	*GetSpaceSuccess 0*	*RequestedSize (16)*	The PMMU successfully satisfied in block 0 of LPM a GetSpace() of *RequestedSize* bytes.
	40	*GetSpaceSuccess 1*	*RequestedSize (16)*	The PMMU successfully satisfied in block 1 of LPM a GetSpace() of *RequestedSize* bytes.
	41	*GetSpaceSuccess 2*	*RequestedSize (16)*	The PMMU successfully satisfied in block 2 of LPM a GetSpace() of *RequestedSize* bytes.
	42	*GetSpaceSuccess 3*	*RequestedSize (16)*	The PMMU successfully satisfied in block 3 of LPM a GetSpace() of *RequestedSize* bytes.
	43	*GetSpaceFail*	*RequestedSize (16)*	The PMMU could not satisfy a GetSpace() of *RequestedSize* bytes.
	44	*GetSpaceFail*	*TotalFreeWords (16)*	The PMMU could not satisfy a GetSpace(). The data event is the total number of words (4 bytes) free in the LPM.
	45	*PacketDeallocation0*	*Block0FreeBytes (16)*	The PMMU de-allocates space in block 0 of the LPM due to a downloading of a packet. The event data is the number of bytes free in the block before the de-allocation occurs.

Fig. 37

PMMU	46	*PacketDeallocation1*	*Block1FreeBytes (16)*	The PMMU de-allocates space in block 1 of the LPM due to a downloading of a packet. The event data is the number of bytes free in the block before the de-allocation occurs.
	47	*PacketDeallocation2*	*Block2FreeBytes (16)*	The PMMU de-allocates space in block 2 of the LPM due to a downloading of a packet. The event data is the number of bytes free in the block before the de-allocation occurs.
	48	*PacketDeallocation3*	*Block3FreeBytes (16)*	The PMMU de-allocates space in block 3 of the LPM due to a downloading of a packet. The event data is the number of bytes free in the block before the de-allocation occurs.
QS	49	*InsertFromPMMU*	*FreeEntriesInQS (8)*	A packet identifier is inserted from the PMMU into one of the queues. The event data is the number of free entries in the pool of entries before the insertion.
	50	*InsertFromCU*	*FreeEntriesInQS (8)*	A packet identifier is inserted from the CU into one of the queues. The event data is the number of free entries in the pool of entries before the insertion.
	51	*InsertFromQS*	*FreeEntriesInQS (8)*	A packet identifier is inserted from the QS into one of the queues. The event data is the number of free entries in the pool of entries before the insertion.
CU	52	*InsertPMMU*	*FreePMMUcmdEntries (4)*	A command is inserted in the PMMU command queue. The event data is the number of free entries in this queue before the insertion.
	53	*InsertQS*	*FreeQScmdEntries (4)*	A command is inserted in the QS command queue. The event data is the number of free entries in this queue before the insertion.

Fig. 38

CU	54	*insertRTU*	*FreeRTUcomdEntries (4)*	A command is inserted in the RTU command queue. The event data is the number of free entries in this queue before the insertion.
	55	*ResponseInsert*	*NumOfResponses (1)*	One or two responses are inserted in the response queue. The event data *NumOfResponses* codes how many (0:one, 1:two).
RTU	56	*Activate*	*NumPMUownedCtx (3)*	A context becomes SPU-owned. The event data is the current number of PMU-owned contexts before the activation.
	57	*PreloadStarts*	*SIUlatency (8)*	A pre-load of a context starts. The event data is the number of cycles (up to 255) that the RTU waited for the first header data to preload is provided by the SIU.
	58	*PreloadAccepted*	*NumOfPreloadsWaiting (3)*	A packet identifier is accepted from the QS. The event data is the number of valid entries in the new packet table before the acceptance.
	59	*CommandWaits*	*CommandWaitCycles (8)*	A command from the CU is ready. The event data is the number of cycles (up to 255) that it waits until it is served.
LPM	60	*ReadSIU*	*SIUwaitCycles (3)*	The SIU performs a read into the LPM. The event data is the number of cycles (up to 7) that it waits until it can be served.
	61	*WriteSIU*	*SIUwaitCycles (3)*	The SIU performs a write into the LPM. The event data is the number of cycles (up to 7) that it waits until it can be served.

Table 1: Events probed for performance counters

Fig. 39

Block	#	Name	Description
IBU	0	*HeadAlwaysValid*	The IBU always provides a valid packet. The packet provided is a 16-byte packet, from devide Id 0, with the 3rd byte 0, and byte *i* (*i*=4..15) to value *i*.
OBU	4	*HeadAlwaysValid*	The OBU always provides a valid packet. The packet provided is a 16-byte packet, from devide Id 0, with the 3rd byte 0, and byte *i* (*i*=4..15) to value *i*.
	5	*AlwaysToDevId0*	The OBU hardwires the outbound device identifier to 0.
	6	*AlwaysToDevId1*	The OBU hardwires the outbound device identifier to 1.
PMM	8	*SimpleAllocation*	The PMM performs the following allocation mechanism when receives a new packet: o 64K bytes (1 full block) are always allocated (i.e. the size of the packet is not taken into account). o One bit per block indicates whether the block is busy (i.e. it was selected to store a packet). The download of that packet resets the bit. o If more that non-busy block exists, the block with the smallest index is chosen. o If no available blocks exist, the packet will be dropped.
QSY	16	*AutomaticCompletion*	Whenever a packet is inserted into a queue (from the PMM or from the SPU), the Complete bit is automatically asserted.
	17	*QueueAlways0*	When a packet is inserted (from any source), the queue will always be queue number 0.
CMU	24	*DummyReplyFromQSY*	Whenever the CMU receives from the SPU a command to the QSY that needs a response back, the CMU generates a dummy response and does not send the command to the QSY. The data associated to the dummy response is 0, and the context number is the same as the one obtained from the SPU.
	25	*DummyReplyFromPMM*	Whenever the CMU receives from the SPU a command to the QSY that needs a response back, the CMU generates a dummy response and does not send the command to the QSY The data associated to the dummy response is 0, and the context number is the same as the one obtained from the SPU

Fig. 40

Architecture block name	Hardware block name
IB	IBU0
OB	OBU0
PMMU	PMM0
LPM	LPM0
QS	QSY0
RTU	RTU0
CU	CU0

Fig. 41

signals are registered by source block unless otherwise specified.

Name	Size	SRC Block	DST Block	Description
Interrupts				
overflowStarted	1	pmm0	exc0	The PMM block decides to store the incoming packet into the EPM.
noMorePagesOfXsize	1	pmm0	exc0	No more virtual pages of the size indicated in the configuration register IntIfNoMoreXsizePages are available
automaticPacketDrop	1	pmm0	exc0	The PMM block cannot store the incoming packet into the LPM and the overflow mechanism is disabled.
packetError	I	pmm0	exc0	Asserted in two cases: The actual packet size received from the external device does not match the value specified in the first two bytes of the packet data. Bus error detected while receiving packet data through the network interface or while downloading packet data from EPM.
lessThanXpacketIdEntries	1	qsy0	exc0	Asserted when the actual number of available entries in the QSY block is less than the value in the configuration register IntIfLessThanXpacketIdEntries.
packetAvailableButNoContext*P*	8 (*P*=0..7)	rtu0	exc0	Asserted when a packet identifier is received by the RTU from the QSY but there is no available context. The level of the interrupt (*P*) depends on how the PMU is configured.
Response Generation				
validResponse	1	cmu0	com0	The CMU has a valid response.
responseData	29	cmu0	com0	The response data.
responseContext	3	cmu0	com0	The context number to which the response will go.
Context Access				
resetContext	1	rtu0	rgf0,rgf1	All GPR registers in context number contextNumber are set to 0.
enableRead0..7	8x1	rtu0	rgf0,rgf1	Read port 0..7 of context number contextNumber is enabled.
enableWrite0..3	4x1	rtu0	rgf0,rgf1	Write port 0..7 of context number contextNumber is enabled.
contextNumber	8	rtu0	rgf0,rgf1	The context number, in 1-hot encoding (LSB bit corresponds to context #0; MSB to context #7) being either read (masked load or pre-load)

Fig. 42

				The context number, in 1-hot encoding (LSB bit corresponds to context #0; MSB to context #7) being either read (masked load or pre-load) or written (masked store). The contextNumber bus needs to have the correct value at least one cycle before the first enableRead or enableWrite signals, and it needs to be de-asserted at least one cycle before the last enableRead or enableWrite signals.
registerToRead0..7	8x5	rtu0	rgf0,rgf1	Index of the register(s) to read through read ports 0..7 in context number contextNumber. Validated with the enableRead0..7 signals.
registerToWrite0..3	4x5	rtu0	rgf0,rgf1	Index of the register(s) to write through write ports 0..3 in context number contextNumber. Validated with the enableWrite0..3 signals.
cluster0readData0..7	8x32	rgf0,rgf1	rtu0	The contents of the register(s) read through read ports 0..7 in cluster 0.
cluster1readData0..7	8x32	rgf0,rgf1	rtu0	The contents of the register(s) read through read ports 0..7 in cluster 1.
writeData0..3	4x32	rtu0	rgf0,rgf1	The contents of the register(s) to write through write port(s) 0..3 into context number contextNumber.
Command Request				
statePMMqueue	1	cmu0	dis0,dis1	If asserted, it indicates that a command will be accepted into the PMM queue.
stateQSYqueue	1	cmu0	dis0,dis1	If asserted, it indicates that a command will be accepted into the QSY queue.
stateRTUqueue	1	cmu0	dis0,dis1	If asserted, it indicates that a command will be accepted into the RTU queue.
validCommandCluster0	1	dis0	cmu0	The command being presented by cluster #0 is valid.
validCommandCluster1	1	dis1	cmu0	The command being presented by cluster #1 is valid.
commandContextCluster0	2	dis0	cmu0	The context number within cluster #0 associated to the command being presented by this cluster.
commandContextCluster1	2	dis1	cmu0	The context number within cluster #1 associated to the command being presented by this cluster.
commandTypeCluster0	2	dis0	cmu0	The type of command being presented by cluster #0 (0:RTU, 1:PMMU, 2:QS).
commandTypeCluster1	2	dis1	cmu0	The type of command being presented by cluster #1 (0:RTU, 1:PMMU, 2:QS).
commandOpcodeCluster0	3	dis0	cmu0	The opcode of the command being presented by cluster #0.
commandOpcodeCluster1	3	dis1	cmu0	The opcode of the command being presented by cluster #1
commandDataCluster0	46	dis0	cmu0	The command data presented by cluster #0.

Fig. 43

commandDataCluster1	46	dis1	cmu0	The command data presented by cluster #1.
Context Unstall				
unstallContext	1	rtu0	cp00	The masked load/store or get context operation performed on context number unstalledContextNum has finished. In case of a get context operation, the misc bus contains the number of the selected context in the 3 LSB bits, and the success outcome in the MSB bit.
preload	1	rtu0	cp00	A pre-load is either going to start (bornContext de-asserted) or has finished (bornContext asserted) on context number unstalledContextNum. The misc bus contains the queue number associated to the packet. If the preload starts and finishes in the same cycle, unstallContext, preload and bornContext are asserted.
bornContext	1	rtu0	cp00	If asserted, the operation performed on context number unstallContextNum is a get context or the end of a pre-load; otherwise it is a masked load/store or the beginning of a pre-load.
unstallContextNum	3	rtu0	cp00	For pre-loads (start or end) it contains the context number of the context selected by the RTU. For get context and masked load/stores, it contains the context number of the context associated to the stream that dispatched the command to the PMU (the RTU receives this context number through the CMU command interface).
misc	30	rtu0	cp00	In case of a pre-load (start or end), it contains the 30-bit code entry point associated to the queue in which the packet resides. In case of a get context operation, the 3 LSB bits contain the selected context number by the RTU, and the MSB bit contains the success bit (whether an available context was found).

unstallContext	preload	bornContext	Action
0	0	0	No operation
0	0	1	Never

Fig. 44

0	1	0	Preload starts
0	1	1	Preload ends
1	0	0	Masked Load/Store ends
1	0	1	GetCtx ends
1	1	0	Never
1	1	1	Preload starts and ends in same cycle

Fig. 45

Signals are registered by source block unless otherwise specified.

Name	Size	SRC Block	DST Block	Description
Network Interface In to the In-Buffer				
dataValue	128	nip0	ibu0	16B of data
validBytes	4	nip0	ibu0	Pointer to the MSB valid byte within dataValue
validData	1	nip0	ibu0	If asserted, at least one byte in dataValue is valid, and validBytes points to the MSB valid byte
rxDevID	1	nip0	ibu0	Device ID of the transmitting device
error	1	nip0	ibu0	Error detected in the current transaction
endOfPacket	1	nip0	ibu0	The current transfer is the last one of the packet
full	1	ibu0	nip0	The buffer in the IBU block is full and it will not accept any more transfers
Network Interface Out from the Out-Buffer (TBD: should the interface be duplicated for each outbound device Id ?)				
dataValue	128	obu0	nop0	16B of data
validBytes	4	obu0	nop0	Pointer to the MSB (if pattern == 0) or to the LSB (if pattern == 1) valid byte in dataValue
pattern	1	obu0	nop0	If pattern == 1 && valid == 0, then no valid bytes. If pattern == 0 && valid == 15, then all 16 bytes are valid
txDevID	1	obu0	nop0	Device ID of the receiving device
err	1	obu0	nop0	Error detected in the current transaction
ready	4	nop0	obu0	Receiving device is ready to accept more data
Overflow Interface to Memory				
dataValue	128	ibu0	ovl0	16B of data
overflowStoreRequest	1	pmm0	ovl0	Initiate an overflow store operation
overflowPageOffset	16	pmm0	ovl0	Offset of the 256B atomic page in the external packet memory
overflowLineOffset	4	pmm0	ovl0	Offset of the first line in the atomic page
extract	1	ovl0	ibu0	Extract the next data from the buffer in the IBU
doneStore	1	ovl0	pmm0	The overflow operation is complete
validBytes	4	ibu0	ovl0	Pointer to the MSB valid byte within dataValue
validData	1	ibu0	ovl0	If asserted, at least one byte in dataValue is valid, and validBytes

Fig. 46

				points to the MSB valid byte
rxDevID	1	ibu0	ovl0	Device ID of the transmitting device
error	1	ibu0	ovl0	Error detected in the current transaction
endOfTransaction	1	ibu0	ovl0	The current transfer is the last one of the transaction
packetSizeMismatch	1	ovl0	pmm0	The SIU detects a packet size mismatch while overflowing a packet.
Overflow Interface from Memory				
dataValue	128	ovl0	obu0	16B of data
validBytes	4	ovl0	obu0	Pointer to the MSB (if pattern == 0) or to the LSB (if pattern == 1) valid byte in dataValue
pattern	1	ovl0	obu0	If pattern == 1 && valid == 0, then no valid bytes. If pattern == 0 && valid == 15, then all 16 bytes are valid
overflowRetrieveRequest	1	pmm0	ovl0	Initiate an overflow retrieve operation
overflowPageOffset	16	pmm0	ovl0	Offset of the 256B atomic page in the external packet memory
overflowLineOffset	4	pmm0	ovl0	Offset of the first line in the atomic page to be used
sizePointer	4	pmm0	ovl0	Offset of the byte in the line that contains the LSB byte of the size of the packet
doneRetrieve	1	ovl0	pmm0	The overflow operation is complete
full0	1	obu0	ovl0	The buffer in the OBU block associated to outbound device identifier 0 is full
full1	1	obu0	ovl0	The buffer in the OBU block associated to outbound device identifier 1 is full
error	1	ovl0	obu0,pmm0	Error detected on the bus as packet was being transferred to outbound device identifier txDevID
txDevID	1	pmm0	ovl0	The outbound device identifier
Local Packet Memory Interface (SPU)				
dataValue	128	lmc0	lpm0	16B of data
dataValue	128	lpm0	lmc0	16B of data
read	1	lmc0	lpm0	Read request. If read is asserted, write should be de-asserted
write	1	lmc0	lpm0	Write request. If write is asserted, read should be de-asserted. When write is asserted, the data to be written should be available in dataValue
dataControlSelect	1	lmc0	lpm0	If asserted, it validates the read or

Fig. 47

				write access
lineAddress	14	lmc0	lpm0	Line number within the LPM to read or write
valid	1	lpm0	lmc0	Access to the memory port (for read or write) is granted
Local Packet Memory/Memory Bus Interface (RTU)				
dataValue	128	lmc0	rtu0	16B of data
dataValue	128	rtu0	lmc0	16B of data
read	1	rtu0	lmc0	Read request. Asserted once (numLines has the total number of 16-byte lines to read)
write	1	rtu0	lmc0	Write request. Asserted on a per-line basis. When asserted, dataValue from RTU should have data to be written
lineAddress	14/32	rtu0	lmc0	Line to initiate access from or to
numLines	4	rtu0	lmc0	Number of lines to read. If numLines == X, then X+1 lines are requested
valid	1	lmc0	rtu0	Access to the operation is granted
backgndStream	1	rtu0	lmc0	Background operation implying only the 14 LSB bits of the line address are used, or streaming operation implying all 32 bits are used
byteEnables	16	rtu0	lmc0	Byte enables. Used only for writing. For reading, byteEnables are 0xFFFF (i.e. all bytes within the all the requested lines are read)
SPU Command Interface through the CMU				
read	1	lmc0	cmu0	Read request. If read is asserted, write should be de-asserted
write	1	lmc0	cmu0	Write request. If write is asserted, read should be de-asserted
dataValue	32	lmc0	cmu0	4B of data
dataValue	32	cmu0	lmc0	4B of data
dataControlSelect	1	lmc0	cmu0	If de-asserted, it validates the read or write access
lineAddress	7	lmc0	cmu0	Address of the configuration register
valid	1	cmu0	lmc0	CMU notifies that dataValue is ready
Performance Counters Interface through the CMU				
eventA	6	????	cmu0	One of the two events (A) requested to be monitored
eventB	6	????	cmu0	One of the two events (B) requested to be monitored
eventDataA	16	cmu0	????	The data associated to event A, if any. This value is meaningful when the corresponding bit in the eventVector is asserted.

Fig. 48

eventDataB	16	cmu0	????	The data associated to event B, if any. This value is meaningful when the corresponding bit in the eventVector is asserted.
eventVector	64	cmu0	????	The event vector (1 bit per event). LSB bit corresponds to event# 0, MSB bit corresponds to event# 63.
On –Chip Instrumentation (OCI) Interface through the CMU				
(TBD)				

Fig. 49

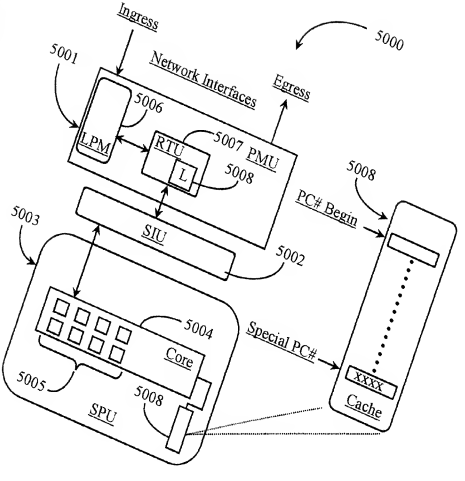

Ingress
5000
5001
Network Interfaces
Egress
5006
LPM
RTU
5007 PMU
L
5008
5008
PC# Begin
5003
SIU
5002
5004
Special PC#
Core
5005
5008
SPU
XXXX
Cache

Fig. 50

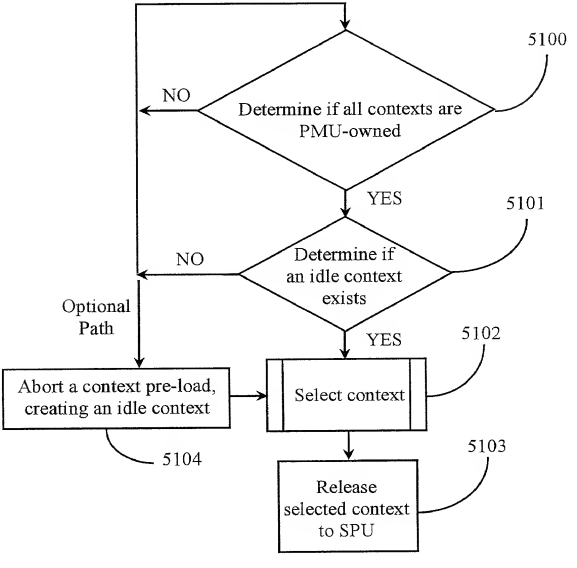


Fig. 51

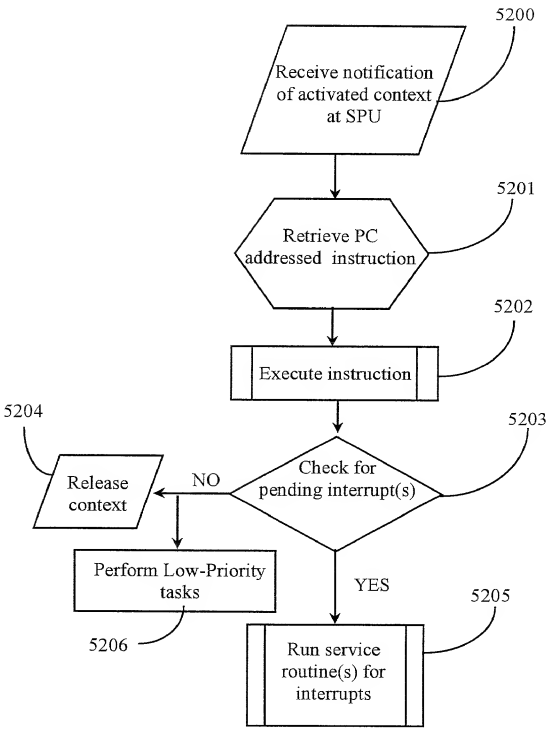

5200
Receive notification of activated context at SPU
5201
Retrieve PC addressed instruction
5202
Execute instruction
5203
Check for pending interrupt(s)
NO
5204
Release context
Perform Low-Priority tasks
5206
YES
5205
Run service routine(s) for interrupts

Fig. 52